

```
// One can also embed points and curves in a volume using the 'Curve/Point In
// Volume' commands:
Extrude {0, 0, 0.1}{ Surface {1}; }

p = newp;
Point(p) = {0.07, 0.15, 0.025, 1c};
Point{p} In Volume {1};

l = newc;
Point(p+1) = {0.025, 0.15, 0.025, 1c};
Line(l) = {7, p+1};
Curve{l} In Volume {1};

// Finally, one can also embed a surface in a volume using the 'Surface In
// Volume' command:
Point(p+2) = {0.02, 0.12, 0.05, 1c};
Point(p+3) = {0.04, 0.12, 0.05, 1c};
Point(p+4) = {0.04, 0.18, 0.05, 1c};
Point(p+5) = {0.02, 0.18, 0.05, 1c};
Line(l+1) = {p+2, p+3};
Line(l+2) = {p+3, p+4};
Line(l+3) = {p+4, p+5};
Line(l+4) = {p+5, p+2};
ll = newcl;
Curve Loop(ll) = {l+1:l+4};
s = news;
Plane Surface(s) = {ll};
Surface{s} In Volume {1};

// Note that with the OpenCASCADE kernel (see 't16.geo'), when the
// 'BooleanFragments' command is applied to entities of different dimensions,
// the lower dimensional entities will be automatically embedded in the higher
// dimensional entities if necessary.

Physical Point("Embedded point") = {p};
Physical Curve("Embdded curve") = {l};
Physical Surface("Embedded surface") = {s};
Physical Volume("Volume") = {1};
```

2.16 t16: Constructive Solid Geometry, OpenCASCADE geometry kernel

See [t16.geo](#). Also available in C++ ([t16.cpp](#)), C ([t16.c](#)), Python ([t16.py](#)), Julia ([t16.jl](#)) and Fortran ([t16.f90](#)).

```

# for a tutorial on model-based views.)

# To create a list-based view one should first create a view:
t1 = gmsh.view.add("A list-based view")

# List-based data is then added by specifying the type as a 2 character string
# that combines a field type and an element shape (e.g. "ST" for a scalar field
# on triangles), the number of elements to be added, and the concatenated list
# of coordinates (e.g. 3 "x" coordinates, 3 "y" coordinates, 3 "z" coordinates
# for first order triangles) and values for each element (e.g. 3 values for
# first order scalar triangles, repeated for each step if there are several time
# steps).

# Let's create two triangles...
triangle1 = [0., 1., 1., # x coordinates of the 3 triangle nodes
             0., 0., 1., # y coordinates of the 3 triangle nodes
             0., 0., 0.] # z coordinates of the 3 triangle nodes
triangle2 = [0., 1., 0., 0., 1., 1., 0., 0., 0.]

# ... and append values for 10 time steps
for step in range(0, 10):
    triangle1.extend([10., 11. - step, 12.]) # 3 node values for each step
    triangle2.extend([11., 12., 13. + step])

# List-based data is just added by concatenating the data for all the triangles:
gmsh.view.addListData(t1, "ST", 2, triangle1 + triangle2)

# Internally, post-processing views parsed by the .geo file parser create such
# list-based data (see e.g. 't7.py', 't8.py' and 't9.py'), independently of any
# mesh.

# Vector or tensor fields can be imported in the same way, the only difference
# being the type (starting with "V" for vector fields and "T" for tensor
# fields) and the number of components. For example a vector field on a line
# element can be added as follows:
line = [
    0., 1., # x coordinate of the 2 line nodes
    1.2, 1.2, # y coordinate of the 2 line nodes
    0., 0. # z coordinate of the 2 line nodes
]
for step in range(0, 10):
    # 3 vector components for each node (2 nodes here), for each step
    line.extend([10. + step, 0., 0.,
                 10. + step, 0., 0.])
gmsh.view.addListData(t1, "VL", 1, line)

# List-based data can also hold 2D (in window coordinates) and 3D (in model
# coordinates) strings (see 't4.py'). Here we add a 2D string located on the
# bottom-left of the window (with a 20 pixels offset), as well as a 3D string
# located at model coordinates (0.5, 0.5, 0):
gmsh.view.addListDataString(t1, [20., -20.], ["Created with Gmsh"])
gmsh.view.addListDataString(t1, [0.5, 1.5, 0.],

```

```

1, 2, 0,
0, 1, 0,
1, 1, 0])

# Note that two additional interpolation matrices could also be provided to
# interpolate the geometry, i.e. to interpolate curved elements.

# Add the data to the view:
gmsh.view.addListData(t2, "SQ", 1, quad)

# In order to visualize the high-order field, one must activate adaptive
# visualization, set a visualization error threshold and a maximum subdivision
# level (Gmsh does automatic mesh refinement to visualize the high-order field
# with the requested accuracy):
gmsh.view.option.setNumber(t2, "AdaptVisualizationGrid", 1)
gmsh.view.option.setNumber(t2, "TargetError", 1e-2)
gmsh.view.option.setNumber(t2, "MaxRecursionLevel", 5)

# Note that the adapted visualization data can be retrived by setting the
# 'returnAdaptive' argument to the 'gmsh.view.getListData()' function.

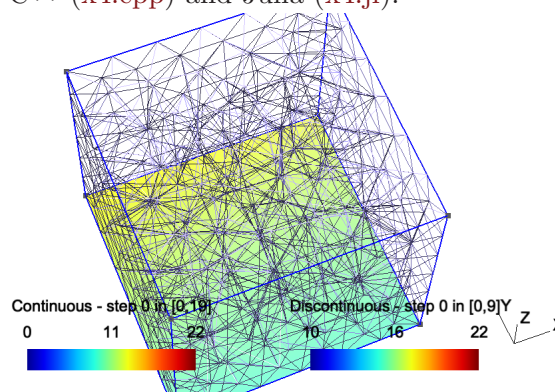
# Launch the GUI to see the results:
if '-nopopup' not in sys.argv:
    gmsh.fltk.run()

gmsh.finalize()

```

2.25 x4: Post-processing data import: model-based

See [x4.py](#). Also available in C++ ([x4.cpp](#)) and Julia ([x4.jl](#)).



```

# -----
#
# Gmsh Python extended tutorial 4
#
# Post-processing data import: model-based
#
# -----

import gmsh
import sys

```

contain formatting characters (%f, %e, etc.). Note that all *expressions* are evaluated as floating point values in Gmsh (see [Section 5.1.2 \[Floating point expressions\]](#), [page 91](#)), so that only valid floating point formatting characters make sense in *string-expression*. See [Section 2.5 \[t5\]](#), [page 26](#), for an example of the use of `Printf`.

`Printf (string-expression , expression-list) > string-expression;`

Same as `Printf` above, but output the expression in a file.

`Printf (string-expression , expression-list) >> string-expression;`

Same as `Printf` above, but appends the expression at the end of the file.

`Warning|Error (string-expression <, expression-list>);`

Same as `Printf`, but raises a warning or an error.

`Merge string-expression;`

Merge a file named *string-expression*. This command is equivalent to the ‘File->Merge’ menu in the GUI. If the path in *string-expression* is not absolute, *string-expression* is appended to the path of the current file. This operation triggers a synchronization of the CAD model with the internal Gmsh model.

`ShapeFromFile(string-expression);`

Merge a BREP, STEP or IGES file and returns the tags of the highest-dimensional entities. Only available with the OpenCASCADE geometry kernel.

`Draw;` Redraw the scene.

`SplitCurrentWindowHorizontal expression;`

Split the current window horizontally, with the ratio given by *expression*.

`SplitCurrentWindowVertical expression;`

Split the current window vertically, with the ratio given by *expression*.

`SetCurrentWindow expression;`

Set the current window by **specifying** its index (starting at 0) in the list of all windows. When new windows are created by splits, new windows are appended at the end of the list.

`UnsplitWindow;`

Restore a single window.

`SetChanged;`

Force the mesh and post-processing vertex arrays to be regenerated. Useful e.g. for creating animations with changing clipping planes, etc.

`BoundingBox;`

Recompute the bounding box of the scene (which is normally computed only after new model entities are added or after files are included or merged). The bounding box is computed as follows:

1. If there is a mesh (i.e., at least one mesh node), the bounding box is taken as the box enclosing all the mesh nodes;
2. If there is no mesh but there is a geometry (i.e., at least one geometrical point), the bounding box is taken as the box enclosing all the geometrical points;
3. If there is no mesh and no geometry, but there are some post-processing views, the bounding box is taken as the box enclosing all the primitives in the views.

This operation triggers a synchronization of the CAD model with the internal Gmsh model.

elementary curves that need to be grouped inside the physical curve. If a *string-expression* is given instead of *expression* inside the parentheses, a string label is associated with the physical tag, which can be either provided explicitly (after the comma) or not (in which case a unique tag is automatically created). In some mesh file formats (e.g. MSH2), specifying negative tags in the *expression-list* will reverse the orientation of the mesh elements belonging to the corresponding elementary curves in the saved mesh file.

5.2.3 Surfaces

Plane Surface (*expression*) = { *expression-list* };

Create a plane surface. The *expression* inside the parentheses is the plane surface's tag; the *expression-list* on the right hand side should contain the tags of all the curve loops defining the surface. The first curve loop defines the exterior boundary of the surface; all other curve loops define holes in the surface. A curve loop defining a hole should not have any curves in common with the exterior curve loop (in which case it is not a hole, and the two surfaces should be defined separately). Likewise, a curve loop defining a hole should not have any curves in common with another curve loop defining a hole in the same surface (in which case the two curve loops should be combined).

Surface (*expression*) = { *expression-list* } < In Sphere { *expression* }, Using Point { *expression-list* } >;

Create a surface filling. With the built-in kernel, the first curve loop should be composed of either three or four curves, the surface is constructed using transfinite interpolation, and the optional **In Sphere** argument forces the surface to be a spherical patch (the extra parameter gives the tag of the center of the sphere). With the OpenCASCADE kernel, a BSpline surface is **constructed** by optimization to match the bounding curves, as well as the (optional) points provided after **Using Point**.

BSpline Surface (*expression*) = { *expression-list* };

Create a BSpline surface filling. Only a single curve loop made of 2, 3 or 4 BSpline curves can be provided. **BSpline Surface** is only available with the OpenCASCADE kernel.

Bezier Surface (*expression*) = { *expression-list* };

Create a Bezier surface filling. Only a single curve loop made of 2, 3 or 4 Bezier curves can be provided. **Bezier Surface** is only available with the OpenCASCADE kernel.

Disk (*expression*) = { *expression-list* };

Creates a disk. When four expressions are provided on the right hand side (3 coordinates of the center and the radius), the disk is circular. A fifth expression defines the radius along Y, leading to an ellipse. **Disk** is only available with the OpenCASCADE kernel.

Rectangle (*expression*) = { *expression-list* };

Create a rectangle. The 3 first expressions define the lower-left corner; the next 2 define the width and height. If a 6th expression is provided, it defines a radius to round the rectangle corners. **Rectangle** is only available with the OpenCASCADE kernel.

Surface Loop (*expression*) = { *expression-list* } < Using Sewing >;

Create a surface loop (a shell). The *expression* inside the parentheses is the surface loop's tag; the *expression-list* on the right hand side should contain the tags

the rotation angle (in radians). With the built-in geometry kernel the angle should be strictly smaller than π .

Extrude { *extrude-list* }

Extrude entities in *extrude-list* using a translation along their normal. Only available with the built-in geometry kernel.

Extrude { *extrude-list* } **Using Wire** { *expression-list* }

Extrude entities in *extrude-list* along the give wire. Only available with the OpenCASCADE geometry kernel.

ThruSections { *expression-list* }

Create surfaces through the given curve loops or wires. **ThruSections** is only available with the OpenCASCADE kernel.

Ruled ThruSections { *expression-list* }

Create ruled surfaces through the given curve loops or wires. **Ruled ThruSections** is only available with the OpenCASCADE kernel.

Fillet { *expression-list* } { *expression-list* } { *expression-list* }

Fillet volumes (first list) on some curves (second list), using the provided radii (third list). The radius list can either contain a single radius, as many radii as curves, or twice as many as curves (in which case different radii are provided for the begin and end points of the curves). **Fillet** is only available with the OpenCASCADE kernel.

Chamfer { *expression-list* } { *expression-list* } { *expression-list* } { *expression-list* }

Chamfer volumes (first list) on some curves (second list), using the provided distance (fourth list) measured on the given surfaces (third list). The distance list can either contain a single distance, as many distances as curves, or twice as many as curves (in which case the first in each pair is measured on the given corresponding surface). **Chamfer** is only available with the OpenCASCADE kernel.

with

extrude-list:

<Physical> Point | Curve | Surface { *expression-list-or-all* }; ...

As explained in [Section 5.1.2 \[Floating point expressions\]](#), [page 91](#), *extrude* can be used in an expression, in which case it returns a list of tags. By default, the list contains the “top” of the extruded entity at index 0 and the extruded entity at index 1, followed by the “sides” of the extruded entity at indices 2, 3, etc. For example:

```
Point(1) = {0,0,0};
Point(2) = {1,0,0};
Line(1) = {1, 2};
out[] = Extrude{0,1,0}{ Curve{1}; };
Printf("top curve = %g", out[0]);
Printf("surface = %g", out[1]);
Printf("side curves = %g and %g", out[2], out[3]);
```

This behaviour can be changed with the `Geometry.ExtrudeReturnLateralEntities` option (see [Section 7.3 \[Geometry options\]](#), [page 252](#)).

5.2.6 Boolean operations

Boolean operations can be applied on curves, surfaces and volumes. All boolean **operation** act on two lists of elementary entities. The first list represents the object; the second represents the tool. The general syntax for boolean operations is as follows:

boolean:

`cornerTags` can be used to specify the (3 or 4) corners of the transfinite interpolation explicitly; specifying the corners explicitly is mandatory if the surface has more **that** 3 or 4 points on its boundary.

Input: `tag` (integer), `arrangement = "Left"` (string), `cornerTags = []` (vector of integers)

Output: -

Return: -

Language-specific definition:

C++, C, Python, Julia

Examples: C++ (`x2.cpp`), Python (`x2.py`, `get_data_perf.py`, `terrain.py`, `terrain_bspline.py`, `terrain_stl.py`)

`gmsh/model/mesh/setTransfiniteVolume`

Set a transfinite meshing constraint on the surface `tag`. `cornerTags` can be used to specify the (6 or 8) corners of the transfinite interpolation explicitly.

Input: `tag` (integer), `cornerTags = []` (vector of integers)

Output: -

Return: -

Language-specific definition:

C++, C, Python, Julia

Examples: C++ (`x2.cpp`), Python (`x2.py`, `terrain.py`, `terrain_bspline.py`, `terrain_stl.py`)

`gmsh/model/mesh/setTransfiniteAutomatic`

Set transfinite meshing constraints on the model entities in `dimTags`, given as a vector of (dim, tag) pairs. Transfinite meshing constraints are added to the curves of the quadrangular surfaces and to the faces of 6-sided volumes. Quadrangular faces with a corner angle superior to `cornerAngle` (in radians) are ignored. The number of points is automatically determined from the sizing constraints. If `dimTag` is empty, the constraints are applied to all entities in the model. If `recombine` is true, the recombine flag is automatically set on the transfinite surfaces.

Input: `dimTags = []` (vector of pairs of integers), `cornerAngle = 2.35` (double), `recombine = True` (boolean)

Output: -

Return: -

Language-specific definition:

C++, C, Python, Julia

Examples: C++ (`x2.cpp`, `x6.cpp`), Python (`x2.py`, `x6.py`)

`gmsh/model/mesh/setRecombine`

Set a recombination meshing constraint on the model entity of dimension `dim` and tag `tag`. Currently only entities of dimension 2 (to recombine triangles into quadrangles) are supported; `angle` specifies the threshold angle for the simple recombination algorithm..

Input: `dim` (integer), `tag` (integer), `angle = 45.` (double)

Output: -

Geometry.CurveType

Display curves as solid color segments (0) or 3D cylinders (1)

Default value: 0

Saved in: `General.OptionsFileName`

Geometry.CurveWidth

Display width of lines (in pixels)

Default value: 2

Saved in: `General.OptionsFileName`

Geometry.DoubleClickedEntityTag

Tag of last double-clicked geometrical entity

Default value: 0

Saved in: -

Geometry.ExactExtrusion

Use exact extrusion formula in interpolations (set to 0 to allow geometrical transformations of extruded entities)

Default value: 1

Saved in: `General.OptionsFileName`

Geometry.ExtrudeReturnLateralEntities

Add lateral entities in lists returned by extrusion commands?

Default value: 1

Saved in: `General.OptionsFileName`

Geometry.ExtrudeSplinePoints

Number of control points for splines created during extrusion

Default value: 5

Saved in: `General.OptionsFileName`

Geometry.FirstEntityTag

First tag (≥ 1) of entities when creating a model

Default value: 1

Saved in: `General.OptionsFileName`

Geometry.FirstPhysicalTag

First tag (≥ 1) of **physical** groups when creating a model

Default value: 1

Saved in: `General.OptionsFileName`

Geometry.HighlightOrphans

Highlight orphan and boundary entities?

Default value: 0

Saved in: `General.OptionsFileName`

Geometry.LabelType

Type of entity label (0: description, 1: elementary entity tag, 2: physical group tag, 3: elementary name, 4: physical name)

Default value: 0

Saved in: `General.OptionsFileName`

Geometry.Light

Enable lighting for the geometry

Default value: 1

Saved in: `General.OptionsFileName`

8 Gmsh mesh size fields

This chapter lists all the Gmsh mesh size fields (see [Section 1.2.2 \[Specifying mesh element sizes\]](#), page 10). Fields can be specified in script files (see [Section 5.3.1 \[Mesh element sizes\]](#), page 113) or using the API (see [Section 6.5 \[Namespace gmsh/model/mesh/field\]](#), page 171). See [Section 2.10 \[t10\]](#), page 37 for an example on how to use fields.

AttractorAnisoCurve

Compute the distance to the given curves and specify the mesh size independently in the direction normal and parallel to the nearest curve. For efficiency each curve is replaced by a set of Sampling points, to which the distance is actually computed.

Options:

CurvesList

Tags of curves in the geometric model

Type: list

Default value: {}

DistMax Maximum distance, above this distance from the curves, prescribe the maximum mesh sizes

Type: float

Default value: 0.5

DistMin Minimum distance, below this distance from the curves, prescribe the minimum mesh sizes

Type: float

Default value: 0.1

Sampling Number of sampling points on each curve

Type: integer

Default value: 20

SizeMaxNormal

Maximum mesh size in the direction normal to the closest curve

Type: float

Default value: 0.5

SizeMaxTangent

Maximum mesh size in the direction **tangeant** to the closest curve

Type: float

Default value: 0.5

SizeMinNormal

Minimum mesh size in the direction normal to the closest curve

Type: float

Default value: 0.05

SizeMinTangent

Minimum mesh size in the direction **tangeant** to the closest curve

Type: float

Default value: 0.5

AutomaticMeshSizeField

Compute a mesh size field that is quite automatic Takes into account surface curvatures and closeness of objects

B Default value: 0
 C Default value: 0
 D Default value: -0.01

ExtractVolume
 Default value: 0

RecurLevel
 Default value: 3

TargetError
 Default value: 0.0001

View Default value: -1

Plugin(CutSphere)

Plugin(CutSphere) cuts the view 'View' with the sphere $(X-X_c)^2 + (Y-Y_c)^2 + (Z-Z_c)^2 = R^2$.

If 'ExtractVolume' is nonzero, the plugin extracts the elements inside (if 'ExtractVolume' < 0) or outside (if 'ExtractVolume' > 0) the sphere.

If 'View' < 0, the plugin is run on the current view.

Plugin(CutSphere) creates one new list-based view. Numeric options:

Xc Default value: 0
 Yc Default value: 0
 Zc Default value: 0
 R Default value: 0.25

ExtractVolume
 Default value: 0

RecurLevel
 Default value: 3

TargetError
 Default value: 0.0001

View Default value: -1

Plugin(DiscretizationError)

Plugin(DiscretizationError) computes the error between the mesh and the geometry. It does so by supersampling the elements and computing the distance between the supersampled points **dans** their projection on the geometry. Numeric options:

SuperSamplingNodes
 Default value: 10

Plugin(Distance)

Plugin(Distance) computes distances to entities in a mesh.

If 'PhysicalPoint', 'PhysicalLine' and 'PhysicalSurface' are 0, the distance is computed to all the boundaries. Otherwise the distance is computed to the given physical group.

19. Could mesh edges/faces be stored in the MSH file?

Edge/faces can be easily generated from the information already available in the file (i.e. nodes and elements), or through the Gmsh API: see tutorial [Section 2.28 \[x7\]](#), [page 75](#), [examples/api/edges.cpp](#) and [examples/api/faces.cpp](#).

C.6 Solver module

1. How do I integrate my own solver with Gmsh?

Gmsh uses the ONELAB interface (<http://www.onelab.info>) to interact with external solvers. See [Section 1.3 \[Solver module\]](#), [page 12](#).

2. Can I launch Gmsh from my solver (instead of launching my solver from Gmsh) in order to monitor a solution?

Using the Gmsh API, you can directly embed Gmsh in your own solver, use ONELAB for interactive parameter definition and modification, and create visualization data on the fly. See e.g. [prepro.py](#), [custom_gui.py](#), [custom_gui.cpp](#).

Another (rather crude) approach [if](#) to launch the Gmsh app everytime you want to visualize something (a simple C program showing how to do this is given in [utils/misc/callgmsh.c](#)).

Yet another approach is to modify your program so that it can communicate with Gmsh through ONELAB over a socket. Select ‘Always listen to incoming connection requests’ in the Gmsh solver option panel (or run gmsh with the `-listen` command line switch), and Gmsh will always listen for your program on the given socket (or on the `Solver.SocketName` if no socket is specified).

C.7 Post-processing module

1. How do I compute a section of a plot?

Use ‘Tools->Plugins->Cut Plane’.

2. Can I save an isosurface to a file?

Yes: first run ‘Tools->Plugins->Isosurface’ to extract the isosurface, then use ‘View->Export’ to save the new view.

3. Can Gmsh generate isovolumes?

Yes, with the CutMap plugin (set the ExtractVolume option to -1 or 1 to extract the negative or positive levelset).

4. How do I animate my plots?

If the views contain multiple time steps, you can press the ‘play’ button at the bottom of the graphic window, or change the time step by hand in the view option panel. You can also use the left and right arrow keys on your keyboard to change the time step in all visible views in real time.

If you want to loop through different views instead of time steps, you can use the ‘Loop through views instead of time steps’ option in the view option panel, or use the up and down arrow keys on your keyboard.

5. How do I visualize a deformed mesh?

Load a vector view containing the displacement field, and set ‘Vector display’ to ‘Displacement’ in ‘View->Options->Aspect’. If the displacement is too small (or too large), you can scale it with the ‘Displacement factor’ option. (Remember that you can drag the mouse in all numeric input fields to slide the value!)

Another option is to use the ‘General transformation expressions’ (in View->Options->Offset) on a scalar view, with the displacement map selected as the data source.